


Article

Comparing the Effectiveness of Scratch and App Inventor with Regard to Learning Computational Thinking Concepts

Youngki Park ¹  and Youhyun Shin ^{2,*}

¹ Department of Computer Education, Chuncheon National University of Education, Chuncheon 24328, Korea; ypark@cnue.ac.kr

² Department of Computer Science and Engineering, Seoul National University, Seoul 08826, Korea

* Correspondence: shinu89@europa.snu.ac.kr

Received: 22 September 2019; Accepted: 30 October 2019; Published: 1 November 2019



Abstract: Scratch and App Inventor are two of the most widely used block-based programming languages for young students. These are educational languages which allow students to program easily by dragging and dropping their code blocks. One question that arises in relation to these educational languages is which of them would be more helpful in fostering computational thinking. It is difficult to answer this question because each language has its own advantages. In this paper, we propose a novel rubric based on Dr. Scratch for assessing both Scratch and App Inventor projects in terms of computational thinking concept learning. We crawled teachers' and students' open and popular projects and automatically calculated their effectiveness scores with regard to learning computational thinking concepts based on our rubric. The experimental results show that (1) Scratch projects scored higher on average in Parallelism, Synchronization and Flow Control, while App Inventor projects scored higher on average in User Interactivity and Data Representation. The results also show that (2) in many cases, large programs with numerous lines of code scored high in all areas of computational thinking concepts.

Keywords: Scratch; App Inventor; computational thinking; educational programming languages

1. Introduction

Scratch [1,2] and App Inventor [3,4] are two of the most widely used block-based programming languages and environments for colleges, universities and K-12 students. As of August of 2019, there are 44,981,198 registered users on the Scratch website and 8,200,000 registered users on the App Inventor website. Both Scratch and App Inventor are educational programming languages that allow novice programmers or even young students to program easily by dragging and dropping their code blocks [5,6]. Although Scratch was originally developed for teaching young students (ages 8–16) [2], it has also been taught at the college level [7]. Similarly, although App Inventor is mainly used in introductory computer science courses for non-majors [3], there are many examples of successful App Inventor courses at the secondary school level [8]. Scratch and App Inventor are both programming languages and environments [2,8].

Students can improve their *computational thinking* [9–14] skills by making applications using these programming languages [1]. The term “computational thinking” was mentioned for the first time by Seymour Papert [15] and has since become widely known through Jeannette Wing’s CACM paper [9]. Computational thinking can be defined as the “thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent” [11]. As Jeannette Wing argued, “Computational thinking is a fundamental skill for everyone, not just for computer scientists” [16].

There have been proposed several rubrics for assessing students' computational thinking skills based on their developed Scratch or App Inventor projects. A typical example is Dr. Scratch [17–19]. The authors of Dr. Scratch redefine *computational thinking concepts* [20] in order to assess students' individual levels of computational thinking. The computational thinking concepts include *Abstraction and Problem Decomposition, Parallelism, Logical Thinking, Synchronization, Flow Control, User Interactivity, and Data Representation*. The authors also presented a rubric which calculates the “computational thinking scores” based on the computational thinking concepts. It is assumed that the higher scores indicate (1) a better understanding of computational thinking concepts and (2) better computational thinking skills acquired.

Our hypothesis is that the degree of improvement in students' computational thinking skills will vary somewhat depending on which programming language they learn. This is in line with the opinion of Edsger W. Dijkstra, who stated “we are all shaped by the tools we train ourselves to use.” Thus, our research questions are as follows:

1. **Which programming language, Scratch or App Inventor, is better for learning each computational thinking concept?**
2. **Do our students need to learn both Scratch and App Inventor in order to enhance their overall computational thinking skills?**

In this paper, we propose a novel rubric based on Dr. Scratch for assessing both Scratch and App Inventor projects in terms of the learning of computational thinking concepts. We examine teachers' and students' popular projects and calculate their “effectiveness scores” in learning computational thinking concepts based on our rubric. To the best of our knowledge, although there have been papers [21–23] comparing Scratch and App Inventor, no papers have compared these languages in terms of computational thinking. Furthermore, although there are rubrics for Scratch projects and App Inventor projects individually, there is no common rubric for assessing both Scratch and App Inventor projects.

The main contributions of our paper are as follows:

- We propose a novel rubric that can be used for assessing both Scratch and App Inventor projects (Section 3).
- We collect open and popular Scratch and App Inventor projects (Section 4.1) and calculate their effectiveness scores with regard to learning computational thinking concepts based on our rubric, thus answering our first research question (Section 4.2).
- We analyze the result of our experiments, thus answering our second research question (Section 4.3).

2. Related Work

Papadakis et al. [22,23] found that Scratch and App Inventor have individual advantages: (1) while we can create web-based programs using Scratch, we can create mobile applications using App Inventor. (2) App Inventor also differs from Scratch in that it can exploit various types of sensors (such as accelerometer sensors and location sensors) without connecting external devices; it can also provide more command/event blocks than Scratch (such as file access blocks or callback functions [24]). (3) In Scratch, we can easily draw images and record sounds, and there are numerous features for young students. For example, we can easily implement parallel execution codes, and while the program code is running, we can visually observe the parts of the code that are executed. (4) On both the Scratch and App Inventor websites, there is a “gallery” where we can share (or “remix”) our projects with other users. However, the Scratch gallery is much more active than the App Inventor gallery. For example, there are some Scratch projects that have received more than ten thousand “hearts.”

Turback et al. [24] noted that depending on which programming language we use, different types of codes can be implemented to create programs that even have the same functionalities. These different types of codes make it challenging to compare Scratch and App Inventor. Figure 1 shows our example

codes for Scratch and App Inventor, which move a sprite every second. While we explicitly use the “wait 1 seconds” block in Scratch, we use the “Timer” event in order to do the same thing in App Inventor.

Moreno-León et al. [18] presented a rubric for Scratch, which they called “Dr. Scratch”. They redefined the “computational thinking concepts” originally proposed by Brennan and Resnick [20]. The redefined computational thinking concepts are classified into the following seven groups: (1) Abstraction and Problem Decomposition, (2) Parallelism, (3) Logical Thinking, (4) Synchronization, (5) Flow Control, (6) User Interactivity, and (7) Data Representation. The advantage of this rubric is that it can be directly (and automatically) applied to Scratch projects.

Sherman et al. [25] presented a rubric for App Inventor called the “App Inventor Project Rubric”. This rubric can assess students’ “mobile computational thinking” skills, and it has the following 14 assessment categories: (1) Screen Interface, (2) Naming, (3) Events, (4) Procedural Abstraction, (5) Globals with Variables or Text Labels, (6) Component Abstraction, (7) Loops, (8) Conditionals, (9) Lists, (10) Data Persistence, (11) Data Sharing, (12) Public Web Services, (13) Accelerometer and Orientation Sensors, and (14) Location Awareness.

Wangenheim et al. [26] presented rubrics for “Snap!” [27] and App Inventor based on earlier work ([18,25] respectively). They also presented a program called “CodeMaster” which automatically calculates the score of each project. However, their rubrics are not integrated into a single rubric, making it difficult to compare projects written in different languages: for example, while the App Inventor rubric includes criteria pertaining to local databases (TinyDB), Snap! does not provide this functionality by default. As another example, while the Snap! rubric includes criteria for Parallel Execution, the App Inventor rubric does not include such criteria.

To the best of our knowledge, there is no unified rubric that can be used to assess both Scratch and App Inventor projects in terms of computational thinking. In Section 3, we present a unified rubric based on the common features of Scratch and App Inventor in order to compare these two languages. Note that although Scratch and App Inventor are both programming languages and environments [2,8], we compare them mostly from the perspective of a programming language, as there are many common features from a language perspective as opposed to an environment perspective.

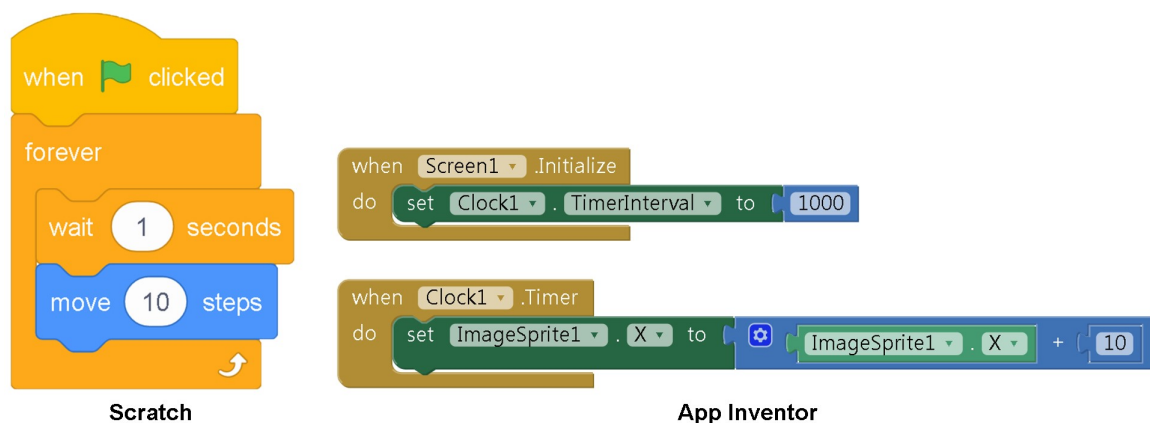


Figure 1. An example of Scratch and App Inventor codes that work in the same way. If we run the Scratch code on the left side of the Figure, the corresponding sprite moves ten steps every second. Likewise, if we run the App Inventor code on the right side of the Figure, the “ImageSprite1” component moves ten steps every second.

3. Unified Rubric for Assessing Both Scratch and App Inventor Projects

One of the barriers to making a unified rubric that can be used for assessing both Scratch and App Inventor is that each programming language has its own code blocks. For example, App Inventor has local file access blocks (TinyDB) while Scratch does not have these types of blocks. Moreover,

while Scratch can create clones of sprites, App Inventor does not have a cloning block. Therefore, we create a novel unified rubric based on the common features of Scratch and App Inventor. We reused the category names of the computational thinking concepts used in Dr. Scratch and modify the detailed criteria by taking into account the common characteristics of Scratch and App Inventor.

Table 1 summarizes the proposed rubric, which assigns a maximum of three points for each computational thinking concept. The detailed criteria of our rubric for assessing each concept are described below.

Table 1. The proposed rubric based on the computational thinking (CT) concepts for Scratch and App Inventor. If a project has at least one element in a particular cell, that project will receive the corresponding score.

CT Concept	Score	Scratch	App Inventor
Abstraction and Problem Decomposition	1	More than one script and more than one sprite	More than one script and more than one component
	2	Definition of blocks	Definition of procedures
	3	Definition of blocks with parameters	Definition of procedures with parameters
Parallelism	1	Two scripts on green flag (1) Two scripts on key pressed	One Clock (1) A Clock enabled by executing code blocks
	2	(2) Two scripts on “when this sprite clicked”	(2) Two different events occurred at the same time
	3	(1) Two scripts on “when I receive message” (2) Two scripts on “when %s > %s”	(1) More than one Clock (2) More than one Image Sprite
		(3) Two scripts on “when backdrop switches to” (4) Two scripts on “when I start as a clone”	(3) More than one Ball
Logical Thinking	1		If statement
	2		If else statement
	3		Logic operations (and/or/not)
Synchronization	1	Wait <i>n</i> seconds	A Clock whose “TimerInterval” property is changed
	2	Broadcast	Screen transitions
	3	Broadcast and wait	Using callback functions
Flow Control	1		Sequence of blocks
	2	(1) Repeat (2) Forever	For each number from, to, by
	3	Repeat until	(1) For each item in list (2) While statement
User Interactivity	1		One or two distinct user events
	2		Three or four distinct user events
	3		More than four distinct user events
Data Representation	1		Modifiers of sprite (component) properties
	2		Operations on variables
	3		Operations on lists

Abstraction and Problem Decomposition. We calculate the Abstraction and Problem Decomposition scores based on the sprites, components, and procedures, because these elements are strongly related to how the program is organized. The “sprite” of Scratch is similar to the “component” of App Inventor. Without a sprite or a component, we do not execute most of the code blocks. Thus, we consider sprites to be the same as the components. We also consider “My Blocks” in Scratch to be identical to “procedures” in App Inventor, as we can define new command blocks with parameters using them. We

did not give additional points to procedures that return a value, because unlike procedures, we cannot return values in My Blocks.

We give one point to projects that have more than one script and more than one sprite (component). If My Blocks or procedures are defined in a project, then the project received two points. We give three points to a project in which parameters are defined.

Parallelism. Parallelism is also one of the core concepts in computational thinking. Thus, if a project has multiple scripts (sequences of code blocks) that run at the same time, that project receives a Parallelism score. In Scratch, we can easily execute multiple command blocks in a parallel manner using the same event blocks in the same sprite. Unlike Scratch, App Inventor does not allow the same event blocks on the same screen. Instead, there are a few other ways to achieve interleaving [24] in App Inventor, such as the use of Clocks, Image Sprites or Balls: (1) we can easily simulate parallel processing in App Inventor using Clocks because the Timer event of a Clock occurs periodically. (2) Additionally, Image Sprites or Balls on the Canvas move consistently in parallel by default.

First, if two scripts begin at the time a program starts, we give one point to that program. Because a Clock is activated by default when its program starts, an App Inventor project receives one point if a Clock is used in it. By the same token, we assign one point if a Scratch project has two green flags in one sprite. Then, we assign two points to projects for which there could be multiple scripts that run at the same time in the middle of the program execution step. For example, when we touch an image sprite in an App Inventor project, both the “when ImageSprite touched” and the “when Canvas touched” events are invoked simultaneously. In this case, we assign two points to the project. Finally, we give three points to projects for which it is possible that one script will affect the behavior of another script due to their parallel execution.

Logical Thinking. The term “Logical Thinking” is closely related to *conditionals* and *logic operators*. Thus, we assign one point for projects that have “if” statements and two points for those that have “if-else” statements. If any of the logic operations (and/or/not) are used in a project, we assign three points to that project. Note we only consider *explicit* conditionals. For example, the GetValue command block of the TinyDB component has the “valueIfTagNotThere” parameter. We use this parameter value if the data that we want to find does not exist. Although this block implicitly exploits the concepts of the “if-else” statement, we do not assign two points for these cases because we are not certain as to whether the code writer intended to use the “else” statement.

Synchronization. Synchronization involves changing the flow of the execution by different scripts, time, or the like. We assign one point to projects that have code blocks for intentionally delaying the execution. We assign two points to the projects that may involve a scene change. Finally, we assign three points to projects that have at least one callback function. While App Inventor has 40 types of callback functions (as of August 2019), Scratch does not support the callback functions. The “Broadcast and wait” block in Scratch is similar to the callback functions in App Inventor. Thus, we assign three points if that block is used at least once. For example, Figure 2 shows a comparison of the callback functions used in Scratch and App Inventor; a sprite says “Cheese!” after another sprite says “No Problem!” (on the left side of the Figure 2), and the background image changes after we take a picture (on the right side of the Figure 2).

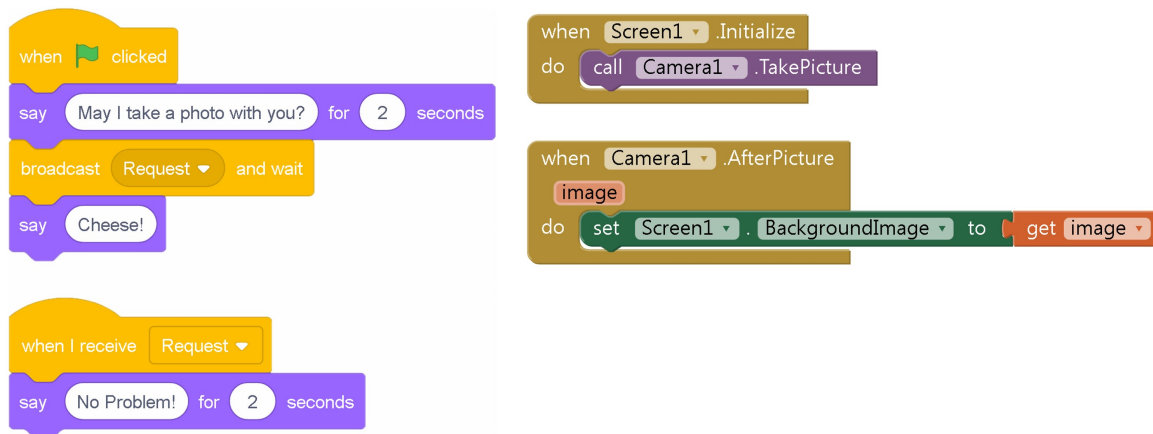


Figure 2. Example codes that have nearly identical structures in different languages related to Synchronization. Comparison of the “Broadcast and wait” statement used in Scratch (on the left side of the figure) and one of the callback functions used in App Inventor (on the right side of the figure).

Flow Control. This type of computational thinking concept is closely related to *loops*. We assign one point to projects that have a sequence of blocks because loops can be simulated by just repeating the statements. We give two points to projects that have simple loops, such as “Repeat,” “Forever” or “For each number from, to, by.” We ignore certain exceptional cases for the sake of simplicity; for example, some Scratch projects have numerous duplicate codes instead of loops, as a “Repeat” block slows down the execution time if Turbo Mode is not used. As another example, some App Inventor projects implement loops using the Clock component and variables. Finally, we assign three points to projects that have more advanced loop blocks.

User Interactivity. Obviously, User Interactivity is strongly related to *events*. While Scratch has seven types of event blocks, App Inventor has 204 types of event blocks. Thus, we decided to assign three points to projects that have more than four *distinct* user events. If there are three or four distinct user events, the project receives two points. A project receives one point if it has only one or two distinct user events.

Data Representation. Data are usually represented in terms of properties, variables or lists. We assign one point to projects that contain at least one block that modifies their properties. If projects define and use variables, then assign them two points. Note some App Inventor programs use the properties of components (mainly Text Labels) to represent variables. We decided to not assign two points to those projects if the variables are not explicitly defined for the sake of simplicity. Finally, if there is a user-defined list in a project, then we assign three points to that project.

4. Experiments

4.1. Experimental Setup

Our goal is to measure the degree to which computational thinking skills can be developed once we have mastered each programming language. In order to achieve our goal, we attempted to crawl mature (and diverse) Scratch and App Inventor projects rather than crawling all public projects, which include many duplicated or temporary projects [28,29].

We crawled 524 high-quality Scratch projects on the Scratch website. More specifically, we crawled (1) 24 “starter projects” as shown on the Ideas menu, which consists of tutorial projects categorized as animations, games, interactive art, music and dance, stories and video sensing; and (2) 500 “popular” and “trending” projects on the Explore menu, which consist of the animations, art, games, music and story projects.

We also crawled 379 mature projects on the App Inventor website, consisting of (1) official tutorials, (2) all of the popular projects “liked” by greater than or equal to ten users, and (3) all of the App of the Month (AOM) projects from January of 2015 to August of 2019. Note that only 209 projects received greater than or equal to ten “likes,” and that 67 AOM projects are developed by adults while the other 89 AOM projects are invented by primary or secondary school students. We excluded projects that could only be downloaded from the Google Play Store because we cannot access to their source codes. We also excluded projects with broken hyperlinks.

Table 2 shows the statistics of our selected projects. The average number of lines of code for each category is also presented in the table. Because all of the selected projects provide their source codes, in Sections 4.2 and 4.3 these codes are parsed using Python and automatically scored according to the proposed rubric.

Table 2. Statistics of the selected projects.

Language	Type	# Projects	Average # Lines
Scratch	Starter Projects	24	37.96
	Animations	50 (popular) + 50 (trending)	602.58
	Art	50 (popular) + 50 (trending)	324.39
	Games	50 (popular) + 50 (trending)	811.85
	Music	50 (popular) + 50 (trending)	508.99
	Stories	50 (popular) + 50 (trending)	456.48
	Total	524	517.82
App Inventor	Official Tutorials	14	21.43
	Gallery (popular)	209 (greater than or equal to 10 likes)	109.12
	App of the Month (youth)	89	314.39
	App of the Month (adult)	67	353.16
	Total	379	197.23

4.2. Experimental Results on Computational Thinking Concepts: Scratch vs. App Inventor

Table 3 shows the overall comparison results for Scratch and App Inventor for computational thinking concepts. In each cell of the second and third columns, the numbers outside of the parentheses are the averages of the effectiveness scores calculated by the proposed rubric, while the values inside the parentheses are the standard deviations of the scores. We also specify their two-tailed p -values in the fourth column in order to determine whether the differences in the two effectiveness scores are statistically significant. Note that the maximum available value of each effectiveness score was three.

Table 3. The overall comparison result of Scratch and App Inventor in terms of computational thinking concepts according to t -test. The numbers outside of the parentheses are the averages of the effectiveness scores calculated by the proposed rubric, while the values inside the parentheses are the standard deviations of the scores.

CT Concept	Scratch	App Inventor	p -Value
Abstraction and Problem Decomposition	1.48 (1.05)	1.49 (0.82)	0.8773
Parallelism	2.43 (0.96)	1.21 (1.29)	0.0001 ***
Logical Thinking	1.61 (1.39)	1.45 (1.14)	0.0664
Synchronization	2.07 (1.01)	1.56 (1.26)	0.0001 ***
Flow Control	2.34 (0.66)	1.24 (0.64)	0.0001 ***
User Interactivity	1.66 (0.65)	2.10 (0.85)	0.0001 ***
Data Representation	1.76 (0.77)	1.88 (0.93)	0.0346 *

* $p \leq 0.05$, *** $p \leq 0.001$.

First, the experimental results show that there are no statistically significant differences between the scores of Scratch and App Inventor in the areas of Abstraction and Problem Decomposition and Logical Thinking. Unfortunately, the scores for these computational thinking concepts are relatively low compared to those of the other concepts. This result is similar to previous findings [28] which showed many Scratch projects do not use My Blocks or conditionals.

With three of the computational thinking concepts (Parallelism, Synchronization and Flow Control), Scratch projects outperform App Inventor projects, especially with regard to Parallelism, where the average score for Scratch is much higher than that for App Inventor. This result is in good agreement with work by Wangenheim et al. [26], who noted that it is difficult to implement parallelism in App Inventor. On the other hand, this result also shows that Scratch is highly appropriate for learning parallelism. Scratch also outperforms App Inventor in terms of Synchronization. This result is somewhat surprising because App Inventor has 40 different types of callback functions. Scratch also scored higher in the area of Flow Control. In practice, while Repeat blocks are frequently used to control sprites in Scratch, it is often unnecessary to use loops when changing the state of a component in App Inventor.

App Inventor projects scored higher in the areas of User Interactivity and Data Representation. The high User Interactivity score may stem from the fact that we often need to use the event blocks of a component when we need to exploit the functionalities of the component. Additionally, the high score for Data Representation may be due to the fact that there are many command blocks that require a list as input.

In summary, the effectiveness scores for Scratch and App Inventor projects showed significant differences. More specifically, Scratch projects outperformed App Inventor projects in the areas of Parallelism, Synchronization and Flow Control. On the other hand, App Inventor projects outperformed Scratch in the areas of User Interactivity and Data Representation.

4.3. Detailed Analysis of the Results

Project Types. For a more detailed analysis, we initially investigated how the scores of computational thinking concepts vary depending on the type of projects. Figure 3 shows the results of this analysis: (1) For Scratch, game projects received the highest overall score, and starter projects had the lowest overall score. These results indicate that the effectiveness scores of programs differed depending on their genre. (2) For App Inventor, App of the Month projects invented by adults received the highest overall score, while popular projects with greater than or equal to 10 “likes” received the lowest overall score. Although the projects developed by adults scored higher than those developed by young students, their differences were not significant. Additionally, the scores of the young students’ works are much higher than those of the official tutorials. It is important to note many App Inventor tutorials are actually college students’ projects, created as these students took App Inventor classes [3]. This indicates that teaching App Inventor from an early age will effectively allow students to learn computational thinking concepts.

It is interesting to note that projects with high overall scores show significant score improvements in the “weak areas” of each programming language. For example, while the Data Representation score for the Scratch starter projects is 1.08, that for the popular game projects is 2.34. Similarly, while the Parallelism score of the App Inventor tutorials is 0.50, that for the App of the Month (adult) projects is 1.67. These results indicate that even if we learn only one programming language, we can develop comprehensive computational thinking skills if we can make good use of that language.

Project Sizes. Figure 4 shows how the average score of each computational thinking concept changes with an increase in the number of lines. Given that the average number of lines for App Inventor projects is less than 200, we plotted the projects with more than 200 lines as “200+.” These results show that the average scores tend to increase as the number of lines of code increases. This indicates that if we teach students to create large programs, their comprehensive computational thinking skills can be

improved. However, there were exceptions where the average score did not increase significantly even if the number of lines of code was increased. For example, the Flow Control score for App Inventor was relatively low, even for large programs. As another example, the User Interactivity score for Scratch did not increase significantly as the number of lines of code was increased. This indicates that each language has its own area of excellence with regard to the learning of computational thinking skills.

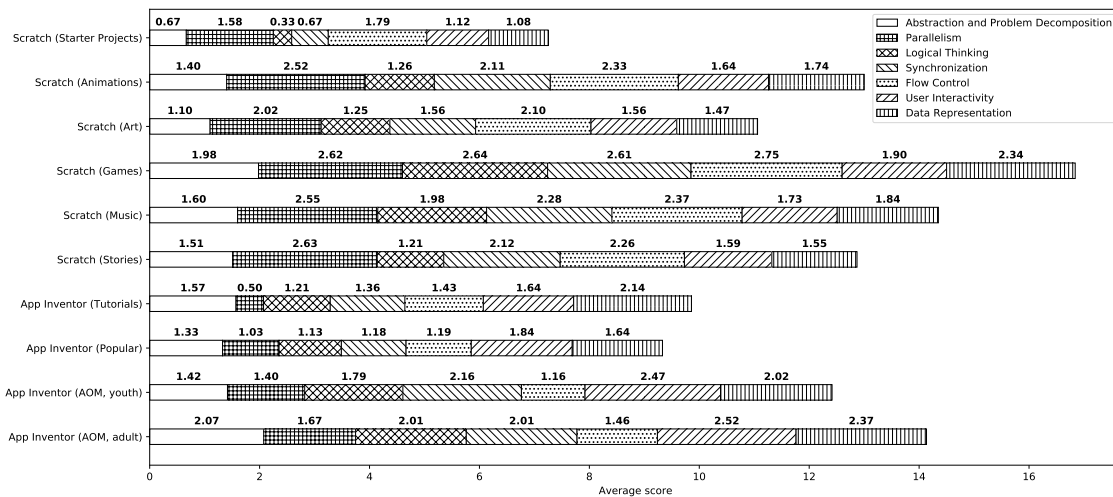


Figure 3. The average score of each computational thinking concept according to the project type. The x-axis is the sum of the mean values of all computational thinking concepts and the y-axis is the program type for each language.

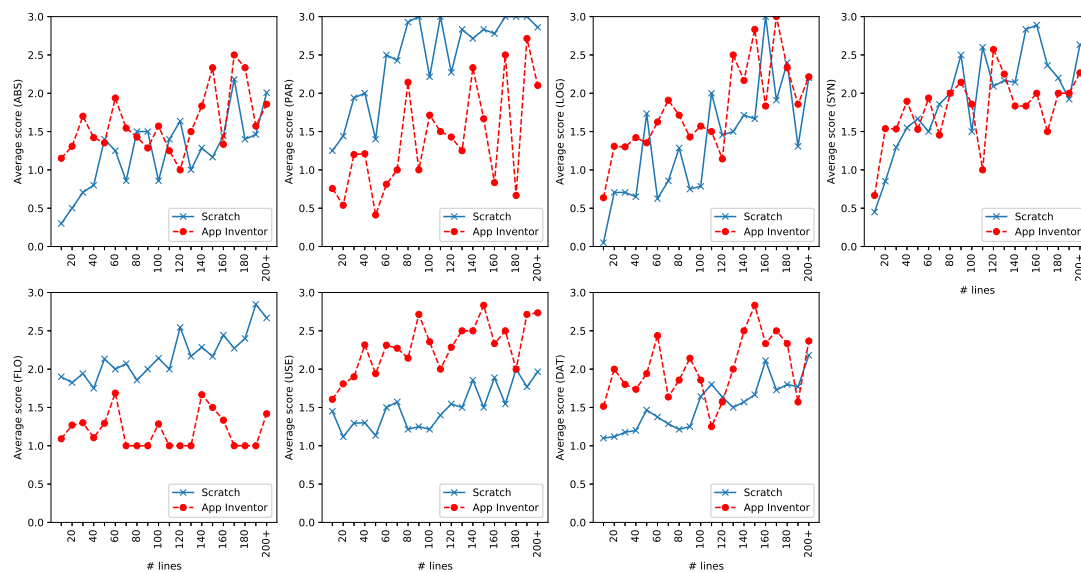


Figure 4. The average scores of each computational thinking concept according to the number of lines of code. ABS, PAR, LOG, SYN, FLO, USE, and DAT correspondingly denote Abstraction and Problem Decomposition, Parallelism, Logical Thinking, Synchronization, Flow Control, User Interactivity, and Data Representation, respectively.

In summary, even if students learn only one programming language, they can develop comprehensive computational thinking skills by making good use of that language. However, each programming language is superior to the other in certain computational thinking areas. Thus, it could

be more helpful for students to learn both Scratch and App Inventor to gain a broader sense of computational thinking concepts.

4.4. Threats of Validity

Although we believe that our findings can be helpful for many computer science educators and K-12 teachers, this paper has the following limitations.

- We experimented with high-quality Scratch and App Inventor projects; therefore, different results may arise for moderate or low quality projects. In other words, this paper assumes that students are sufficiently skilled in the use of each programming language.
- We could not find a large number of high-quality projects on the App Inventor and Scratch websites because (1) App Inventor projects are not actively uploaded or shared in the projects gallery, and (2) on the Scratch website, there are many empty or duplicate Scratch projects [23], making it difficult to crawl high-quality projects. Although we could show statistically significant differences between Scratch and App Inventor regarding the five computational thinking concepts using a total of 903 projects, it would be possible to produce more meaningful results if more projects could be collected.
- When creating the unified rubric, only the common characteristics of Scratch and App Inventor were taken into account. In other words, the rubric does not take into account the unique characteristics of Scratch and App Inventor. For example, file access blocks are excluded from the proposed rubric because these are not supported in Scratch. Further research is needed to analyze how these unique blocks can help improve students' computational thinking skills.

5. Conclusions

In this paper, we compared the effectiveness of Scratch and App Inventor with regard to the learning of computational thinking concepts. In order to compare these languages, we initially proposed a novel rubric for assessing both Scratch and App Inventor projects in terms of computational thinking concept learning. Based on the rubric, we calculated the effectiveness scores of high-quality Scratch and App Inventor projects. Note while some projects were created by teachers or adults, most were made by young students, by themselves. The experimental results show that the effectiveness scores for Scratch and App Inventor projects are significantly different. More specifically, **Scratch projects scored higher on average on Parallelism, Synchronization, and Flow Control, while App Inventor projects scored higher on average on User Interactivity and Data Representation.** However, we also found that the overall scores for programs tend to increase as the number of lines of code increases and that the scores are highly dependent on the types (genres) of projects. Based on these results, we can conclude that **if students learn both Scratch and App Inventor, they can improve their computational thinking skills in general, and that mastering a single programming language can also improve their overall computational thinking skills if they practice creating various types of large programs.**

Our future work will take three directions. First, we would like to survey or interview groups of experts (such as teachers and computer scientists) with regard to the learning of computational thinking concepts through Scratch and App Inventor. It would be very useful to uncover the difference between the results obtained by selected projects (artefacts) and the results obtained through expert groups. Second, we want to prepare educational materials for these languages in order to develop students' comprehensive computational thinking skills based on the rubric proposed in this study. In particular, we would like to focus on what types of projects could be taught to students in order to foster computational thinking more effectively. Finally, we want to implement a more general rubric that consists not only of block-based programming languages but also of text-based programming languages such as Python and JavaScript, as these text-based programming languages are also widely used for educational purposes.

Author Contributions: Conceptualization, Y.P. and Y.S.; data curation, Y.P.; investigation, Y.P. and Y.S.; methodology, Y.P. and Y.S.; supervision, Y.S.; validation, Y.P. and Y.S.; writing—original draft, Y.P. and Y.S.; writing—review and editing, Y.P. and Y.S..

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Milner A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. Scratch: Programming for all. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]
2. Maloney, J.; Resnick, M.; Rusk N.; Silverman, B.; Eastmond, E. The Scratch programming language and environment. *ACM Trans. Comput. Educ.* **2010**, *10*, 16. [CrossRef]
3. Wolber, D. App inventor and real-world motivation. In Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX, USA, 9–12 March 2011; pp. 601–606.
4. Wolber, D.; Abelson, H.; Spertus, E.; Looney, L. *App Inventor*; O'Reilly Media: Newton, MA, USA, 2011.
5. Xie, B.; Shabir, I.; Abelson, H. Measuring the usability and capability of app inventor to create mobile applications. In Proceedings of the 3rd International Workshop on Programming for Mobile and Touch, Pittsburgh, PA, USA, 27–27 October 2015; pp. 1–8.
6. Liu, J.; Wimmer, H.; Rada, R. Hour of Code: Can it change students' attitudes toward programming? *J. Inf. Technol. Educ. Innov. Pract.* **2016**, *15*, 53–73.
7. Rizvi, M.; Humphries, T.; Major, D.; Jones, M.; Lauzun, H. A CS0 course using Scratch. *J. Comput. Sci. Coll.* **2011**, *26*, 19–27.
8. Wolber, D.; Abelson, H.; Friedman, M. Democratizing Computing with App Inventor. *GetMobile Mob. Comput. Commun.* **2015**, *18*, 53–58. [CrossRef]
9. Wing, J.M. Computational thinking. *Commun. ACM* **2006**, *49*, 33–35. [CrossRef]
10. Wing, J.M. Computational Thinking and thinking about computing. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2008**, *366*, 3717–3725. [CrossRef] [PubMed]
11. Wing, J.M. Research Notebook: Computational Thinking—What and Why. 2011. Available online: <http://people.cs.vt.edu/~kafura/CS6604/Papers/CT-What-And-Why.pdf> (accessed on 31 October 2019).
12. Grover, S.; Pea, R. Computational thinking in K-12: A review of the state of the field. *Educ. Res.* **2013**, *42*, 38–43. [CrossRef]
13. Yadav, A.; Mayfield, C.; Zhou, N.; Hambrusch, S.; Korb J.T. Computational thinking in elementary and secondary teacher education. *ACM Trans. Comput. Educ.* **2014**, *14*, 5. [CrossRef]
14. Selby, C.; Woollard, J. Computational Thinking: The Developing Definitions. 2013. Available online: <https://eprints.soton.ac.uk/356481/> (accessed on 31 October 2019).
15. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books, Inc.: New York, NY, USA, 1980.
16. Barr, D.; Harrison, J.; Conery, L. Computational thinking: A digital age skill for everyone. *Learn. Lead. Technol.* **2011**, *38*, 20–23.
17. Moreno-León, J.; Robles, G.D. Scratch: A Web Tool to Automatically Evaluate Scratch Projects. In Proceedings of the Workshop in Primary and Secondary Computing Education, London, UK, 9–11 November 2015; pp. 132–133.
18. Moreno-León, J.; Robles, G.; Román-González, M.D. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia* **2015**, *46*, 1–23.
19. Moreno-León, J.; Robles, G. Analyze your Scratch projects with Dr. Scratch and assess your computational thinking skills. In Proceedings of the Scratch Conference 2015, Amsterdam, The Netherlands, 12–15 August 2015; pp. 12–15.
20. Brennan, K.; Resnick, M. New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, BC, Canada, 13–17 April 2012.
21. Nikou, S.A.; Economides, A.A. Transition in student motivation during a scratch and an app inventor course. In Proceedings of the 2014 IEEE Global Engineering Education Conference, Istanbul, Turkey, 3–5 April 2014; pp. 1042–1045.

22. Papadakis, S.; Kalogiannakis, M.; Zaranis, N.; Orfanakis, V. Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study. *Int. J. Technol. Enhanc. Learn.* **2016**, *8*, 217–233. [CrossRef]
23. Papadakis, S.; Kalogiannakis, M.; Orfanakis, V.; Zaranis, N. Novice programming environments. Scratch App Inventor: A first comparison. In Proceedings of the 2014 Workshop on Interaction Design in Educational Environments, Albacete, Spain, 9 June 2014.
24. Turbak, F.; Sherman, M.; Martin, F.; Wolber, D.; Pokress S.C. Events-first programming in App Inventor. *J. Comput. Sci. Coll.* **2014**, *29*, 81–89.
25. Sherman, M.; Martin, F.; Baldwin, L.; DeFlippo, J. App Inventor Project Rubric-Computational Thinking through Mobile Computing. 2014. Available online: <https://nsfmobilect.files.wordpress.com/2014/09/mobile-ct-rubric-for-app-inventor-2014-09-01.pdf> (accessed on 31 October 2019).
26. Wangenheim, C.G.; Hauck, J.C.R.; Demetrio, M.F.; Pelle, R.; Alves, N.C.; Azevedo, L.F.; Barbosa, H. CodeMaster-Automatic Assessment and Grading of App Inventor and Snap! Programs. *Inform. Educ.* **2018**, *17*, 117–150. [CrossRef]
27. Harvey, B.; Garcia, D.; Paley, J.; Segars, L. Snap!: (build your own blocks) (abstract only). In Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, Raleigh, NC, USA, 29 February–3 March 2012; p. 662.
28. Aivaloglou, E.; Hermans, F. How kids code and how we know: An exploratory study on the Scratch repository. In Proceedings of the 2016 ACM Conference on International Computing Education Research, Melbourne, Australia, 8–12 September 2016; pp. 53–61.
29. Okerlund, J.; Turbak, F. A preliminary analysis of app inventor blocks programs. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, San Jose, CA, USA, 15–19 September 2013; pp. 15–19.



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).